Overview

Resources allow users represent and reason about a wide range of resource types, including reusables(eg. a machine) and reservoirs(eg. a battery). Individual tokens (activities) can produce or consume resources (or both). Resources can have upper and lower limits, as well as bounds on instantaneous production/consumption and cumulative production/consumption.

Reasoning about resources involves two pieces:

1. **Profile:** The Profile computes the upper and lower bounds for the resource quantity over time, resulting in a resource *envelope*. Also included is information about max/min production and consumption (both instantaneous and cumulative).
2. **Flaw and Violation Detector (FVDetector):** An FVDetector decides how to report flaws and violations given the resource profiles. A violation indicates that decisions have been made that make it impossible to satisfy the resource constraints. A flaw indicates that some choices allowed by the profile will violated constraints (ie things aren't yet broken, but still could become broken).

There are a variety of Profile and FVDetector implementations available, and it is straightforward to implement your own.

To include resources in your model usually involves these steps:

1. In NDDL, extend existing resource classes to get desired behavior:
    1. *Reservoir* resources can have both 'consume' and 'produce' tokens. Note that these tokens do **NOT** have the usual start/end/duration token variables, since occur at a single instant in time. Instead, they have a single variable, 'time' used to represent that instant.
    2. *Reusable* resources have 'uses' tokens that use a quantity for the duration of the token. These tokens do have the usual start/end/duration variables.
    3. *Unary* resources are reusable resources with unit quantity and have 'use' tokens (again, with the usual start/end/duration variables).
2. In NDDL, specify what type of profile is used to represent maximum/minimum values over time (default is IncrementalFlowProfile).
3. In NDDL, specify what type of detector is used to report flaws and violation (default is !ClosedWorldFVDetector).
4. In the planner configuration file specify how resource threats and unbound variables should be decided. The default file (PLASMA/trunk/config/PlannerConfig.xml) is reasonable for most cases.

For example, here is a user-defined resource that extends the Reservoir resources and uses the grounded version of the Profile and FVDetector:

```
class Battery extends Reservoir {
        string profileType;
        string detectorType;

        Battery()
        {
                super(0.0 ,0.0, 10.0); // initial, lower_limit, upper_limit
                profileType = "GroundedProfile";
                detectorType = "GroundedFVDetector";
        }
}
```

The following snippet configures the threat manager to ignore threats on the above Battery resource. Without this line, the built-in solver gets stuck because there is no built-in way to solve these threats. However, the 'Reservoir' filter in the default PlannerConfig.xml would also work:

```
<ThreatManager defaultPriority="0">
  <FlawHandler component="StandardThreatHandler"/>
  <FlawFilter class-match="Battery"/>
</ThreatManager>
```

Finally, the resource lines in the following configuration snippet are not required, but are usually what you want (and is included in the default PlannerConfig.xml):

```
<UnboundVariableManager defaultPriority="0">
  <FlawFilter var-match="start"/>
  <FlawFilter var-match="end"/>
  <FlawFilter var-match="duration"/>
  <FlawFilter class-match="Resource" var-match="time"/>
  <FlawFilter class-match="Resource" var-match="quantity"/>
  <FlawFilter class-match="Reservoir" var-match="time"/>
  <FlawFilter class-match="Reservoir" var-match="quantity"/>
  <FlawFilter class-match="Reusable" var-match="quantity"/>
  <FlawFilter component="InfiniteDynamicFilter"/>
  <FlawHandler component="StandardVariableHandler"/>
</UnboundVariableManager>
```

NEW: Resources as Constraints

This pages focuses on the traditional way to represent resources in EUROPA, which involved a token for each usage of a resource. EUROPA now includes functionality to represent directly with a constraint, without the need for an additional token. This reduces a lot of overhead computations that both propagate changes through tokens, and also reduces the number of decisions that need to be made, since there are no tokens to activate, etc.

TODO: Javier, explain how this works and the pros/cons involved. Provide example.

Common Problems

- **The solver fails to find solutions to a simple problem:** Be sure the configuration file includes resource filters as described above.
- **Profile computations are too slow:** Will GroundedProfile work for you?
- **Can't use start/end/duration for reservoir tokens:** Reservoir tokens (consume and produce) only have 'time' as a variable rather than start/end/duration because unlike other tokens, they occur at a single instant in time.

Options

The following Profile variants are available in EUROPA:

- **IncrementalFlowProfile:** Computes the tightest possible envelope (upper and lower resource bounds). Note that this should not be used if there are limits on instantaneous/cumulative production/consumption as

the relevant numbers are not computed by this profile
- **FlowProfile** (deprecated): An early implementation of IncrementalFlowProfile that is not as efficient, but should produce identical results.
- **TimetableProfile**: A fast profile computation that results in a very loose envelope (upper and lower resource bounds). The upper bounds are computed by assuming all production occurs as early as possible and all consumption occurs as late as possible, and vice versa for the lower bounds. No temporal constraints are considered when these bounds are considered. In many domains, this will profile will result in phantom flaws (flaws due to the loose bounds that cannot be resolved without specifying start/end times).
- **GroundedProfile**: A fast profile computation that can be combined with GroundedFVDetector to only report flaws or violations in the early-start schedule. The envelope is computed assuming all production and consumption occur as early as possible.

The following FVDetector variants are available in EUROPA:

- **ClosedWorldFVDetector**: A 'pessimistic' detector that:
    1. Reports violations if the envelope ever falls completely outside the lower/upper limits (for example, if the lower bound of the envelope exceeds the upper limit). The assumption is that all relevant transactions already exist, and result in an inconsistent state.
    2. Reports flaws if the envelope ever falls partially outside the lower/upper limits (for example, if the lower bound of the envelope falls below the lower limit).
- **OpenWorldFVDetector**: An 'optimistic' detector that:
    1. Reports violations if the envelope ever falls completely outside the lower/upper limits AND added production/consumption cannot remedy the situation. Note that this means violations will only be reported if there are limits on instantaneous and/or cumulative production and/or consumption (otherwise, added production/consumption can remedy any situation!).
    2. Reports flaws if the envelope ever falls partially outside the lower/upper limits AND choosing appropriate quantities cannot remedy the situation. For example, suppose we have a reusable resource with 10 units available and a single activity, 'A', using the resource. If the quantity needed by 'A' is [5,20], a flaw will not be reported (unlike for the ClosedWorldFVDetector) because we could choose any quantity less than or equal to 10 to remedy the situation.
- **GroundedFVDetector**: Used in conjunction with GroundedProfile, a hybrid detector that assumes the user cares about the early-start schedule (all transactions as early as possible) (See /trunk/src/PLASMA/Resource/component/SAVH_GroundedProfile.hh for more details):
    1. Reports violations the same way as ClosedWorldFVDetector. This is because we do not want a solver to be faced with violations when a non-early start schedule is in fact acceptable.
    2. Reports flaws if the ealy-start schedule results in an envelope that falls partially outside the lower/upper limits.

Implementation Matrices

There are many possible pieces of data that can be computed by profiles and monitored by flaw/violation detectors. Here are the values computed by the profiles:

Here we show which ones are computed and monitored by the various profiles and detectors:

	TimetableProfile	GroundedProfile	FlowProfile	IncrementalFlowProfile
LowerLevelMin	Y	*(1)	Y	Y

LowerLevelMax	Y	*(1)	*(2)	*(2)
UpperLevelMin	Y	*(1)	*(2)	*(2)
UpperLevelMax	Y	*(1)	Y	Y
InstConsumptionMin	Y	*(3)	N	N
InstConsumptionMax	Y	*(3)	N	N
InstProductionMin	Y	*(3)	N	N
InstProductionMax	Y	*(3)	N	N
CumConsumptionMin	Y	*(3)	N	N
CumConsumptionMax	Y	*(3)	N	N
CumProductionMin	|Y	*(3)	N	N
CumProductionMax	|Y	*(3)	N	N

Here are the values monitored by the FVDetectors:

	ClosedWorldFVDetector	OpenWorldFVDetector	GroundedFVDetector
LowerLevelMin	Y	Y	*(1)
LowerLevelMax	N	Y	*(1)
UpperLevelMin	N	Y	*(1)
UpperLevelMax	Y	Y	*(1)
InstConsumptionMin	Y	Y	*(3)
InstConsumptionMax	Y	Y	*(3)
InstProductionMin	Y	Y	*(3)
InstProductionMax	Y	Y	*(3)
CumConsumptionMin	Y	Y	*(3)
CumConsumptionMax	Y	Y	*(3)
CumProductionMin	Y	Y	*(3)
CumProductionMax	Y	Y	*(3)

***(1):** As described in the code, the four levels are used in a non-standard way. This is why the GroundedProfile MUST be used with the GroundedFVDetector.

***(2):** The flow profiles use the LowerLevelMin value for LowerLevelMax and the UpperLevelMax value for UpperLevelMin.

***(3):** The consumption/production computations for the grounded case are not yet grounded. For now, they are computed and reported in the same way as TimetableProfile combined with either of the other detectors. See possible new features below.

Possible New Features

Eventually, we hope to incorporate the following improvements (and bug fixes) into a future version of the Resources module:

- Non-constant upper/lower limits. For example, consider a pool of available cars that might get smaller (cars break) or larger (new cars bought) over time. The only way to represent this currently is with 'dummy' production/consumption events.

- Preferred value version of grounded profiles, so a preferred value (instead of the earliest value) could be used for grounding.
- A state resource, both for unary states (eg: on/off) and multi-state (eg: red/yellow/green). If you need a state resource immediately, we describe how to fake a unarey state resource using the existing framework here.
- The GroundedProfile does not treat instantaneous/cumulative production/consumption as 'grounded' but should.
- Re-architect flaw/violation detection so a user can pick and choose. For example, the closed-world assumption might be desired for violations, but not for flaws.
- The OpenWorldFVDetector treats flaws in a way that is not really related to the 'open-world' concept (it doesn't report flaws due to quantity flexibility). This behavior should be separated out; not necessary as part of open-world approach, and available in closed-world approach.
- Currently, you must know *a priori* whether a token will produce or consume a resource (the implementation assumes the quantity associated with any given token will be positive, and a token is marked as either a consumer or producer). We used to have 'change' tokens whose quantity could be negative or positive. Such a generalization might be helpful in some domains, although we have not yet needed it in practice.

If you have a need for one of these listed features, please contact the EUROPA development team and we will attempt to fast-track support for that features.